Powerful Equivalence Checking in the Bank Supply Process

Giuseppe De Ruvo, Antonella Santone
Department of Engineering, University of Sannio, Benevento, Italy
e-mail: {gderuvo@unisannio.it, santone@unisannio.it}

Domenico Raucci
Department of Economics, University of Chieti-Pescara, Italy
e-mail:d.raucci@unich.it

***Abstract*—Equivalence checking is a powerful formal technique to improve the quality of computer and software systems. It is usually employed to verify the correctness in a model-based design. Notwithstanding, a detailed and precise specification is required in order to apply equivalence checking to a given domain. Unfortunately, certain fields of application like business process management lack of such adequate information. We explore the applicability of equivalence checking to validation of Business Processes described by the aid of Workflow Management systems. Due to the state explosion problem, formal methods are not very popular in the business domain. In fact, the state space grows exponentially in the number of concurrent processes leading to an impracticable verification. In this paper we deal with a heuristic-based methodology developed to beat the state explosion problem when checking non-equivalence. Our contribution is two-fold: (i) we show how equivalence checking can successfully operate in the business modelling and analysis context; (ii) we model and use the bank supply process as a real case study to evaluate and test the heuristic-based methodology. We show and debate encouraging experimental results comparing them with a state of the art model checker i.e. CADP. This suggests that the business community, mostly in the banking field, can take advantage from our efficient methodology based on process algebra.**

***Keywords*-Bussiness Process Management; Formal Methods; CCS; Workflow Verification; Banking Process.**

I. Introduction

Several techniques for formal verification have been developed over the past three decade among them equivalence checking [1]. Equivalence checking is the process of determining whether two systems are equivalent to each other according to some mathematically defined notion of equivalence. Equivalence checking is typically used to verify if a system design conforms to its high-level service specification. Although equivalence checking is currently not applicable to all domains, it is useful for certain restricted fields of application. One of this domain is business process management. In particular, we examine the applicability of equivalence checking to validation of Business Processes that are mapped through the systems of Workflow Management. However, the usage of this formal method in the domain of business process management is still not widely used [2]. This is due also to the state explosion problem, which says that the state space grows exponentially in the number of concurrent processes. Indeed, the parallelism between the processes of the system leads to a number of reachable states which may become very large, in some cases on the order of millions or billions of states. When the number of states is too large to fit in the main memory of a computer, verification quickly breaks down. In the business process taken into account, we came across the state explosion matter. Specifically, this problem has emerged as the result of modelling the bank supply process, in which a not very excessive number of processes in parallel has made difficult the verification using standard model checkers.

Many approaches have been built up to deal with the state explosion problem and reducing the number of the states (see Section V). In this paper to combat the state explosion problem for checking process equivalence we consider an efficient procedure, based on heuristic search [3], proposed in [4]. The procedure is applied to processes described by means of a very compact specification language, the Calculus of Communicating Systems (CCS) [1] defined by Milner. CCS is one of the most well known process algebras and it is largely used for designing concurrent and distributed systems. The approach uses a heuristic function that suggests to expand first the states that offer the most promising way to deduce that two systems are not equivalent. This helps to avoid the exhaustive exploration of the global state graph of the two systems when they are not equivalent. One of the authors of this paper has contributed to the design of Grease (GREedy Algorithm for System Equivalence), a C++ tool supporting the heuristic approach to check non-equivalence of CCS processes. The use of Grease on a sample of small CCS processes has shown a significant reduction of both state-space size and time, compared to traditional equivalence checking algorithms. In order to evaluate the real performance of our tool we had to specify the target process by using LOTOS [5] too. In fact, we compared the results of Grease with the ones achieved by CADP [6], a state of the art model checker, which uses LOTOS as specification language. CADP has been chosen since it is a well-known process algebra formal verification tool with inside an equivalence checker and it is a mature tool which is regularly improved. In fact, for what concerns equivalence checking, CADP offers BISIMULATOR which is an equivalence checker, which takes as input two graphs to be compared (one represented implicitly using the OPEN/CAESAR environment, the other represented explicitly

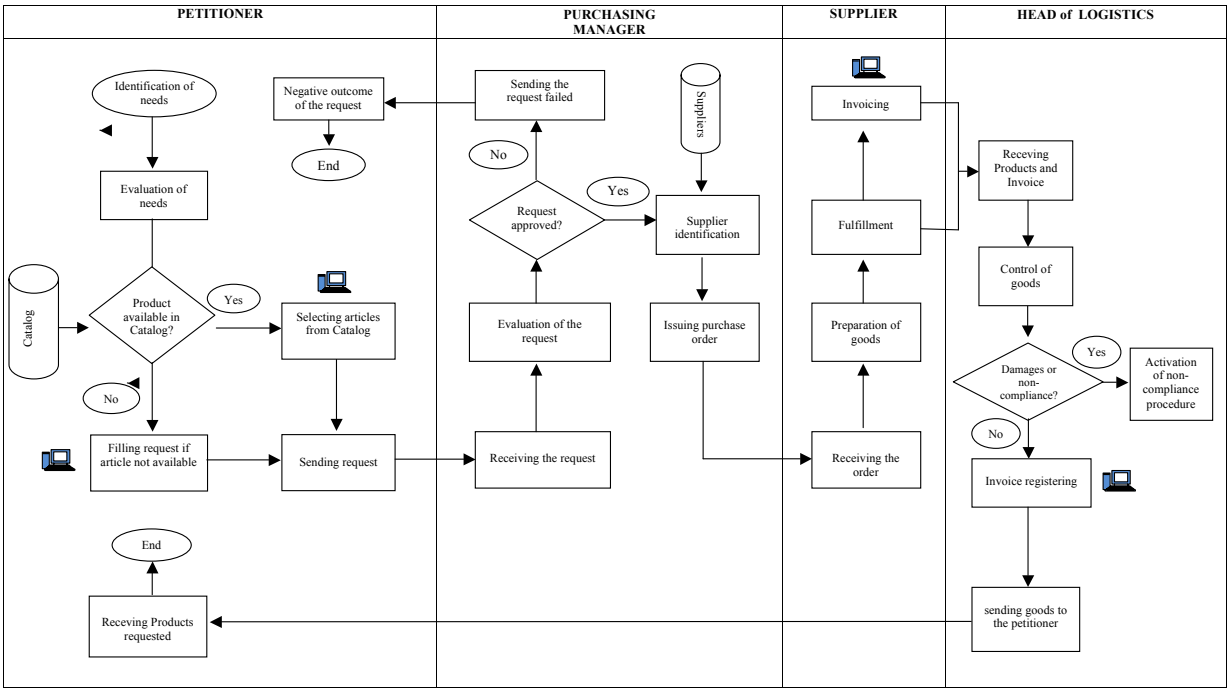

Fig. 1: Bank supply workflow.

as a BCG file) and determines whether they are equivalent (modulo a given equivalence relation). BISIMULATOR works on-the-fly, meaning that only those parts of the implicit graph pertinent to verification are explored. The behavior of our heuristic algorithm is very related to on-the-fly equivalence checking algorithms. However, our approach adds the guidance to the search using heuristic functions.

Thus, our proposal is two-fold: (i) we show how non-equivalence checking can successfully operate in the business modelling and analysis context; (ii) we model and use the bank supply process as a case study to evaluate and test the heuristic-based methodology.

More precisely, in this paper we describe in CCS (resp. LOTOS) both the implementation of the bank supply process and the specifications of the expected behaviours. By applying our approach we can prove the correctness of the system obtaining especially a considerable reduction of execution time but also a reduction of the amount of memory allocation with respect to CADP. Our investigations suggest that the business community, especially in the banking field, can benefit from this efficient methodology developed in the process algebra area to prevent significant errors.

The remainder of the paper is organized as follows. First, in the next Section we give basic definitions of the bank supply process, CCS, LOTOS and a brief rationale on equivalence checking. Section III deals with the heuristic-based methodology. We discuss experimental results in Section IV. Section V summarises related work. Finally, in Section VI we provide conclusions and new inputs for further research.

II. Preliminaries

In this section, after a brief overview of the bank supply process, we present the two process algebras Calculus of Communicating Systems (CCS) [1] and Language of Temporal Ordering Specification [5], and a basic rationale of equivalence checking [1]. In particular, we precisely present CCS, while LOTOS is explained by pointing out the differences with CCS.

A. Bank Supply Process Overview

Formal techniques and tools have a number of features that make it possible to extend their use to specific business domains, including the formal verification of business processes that are mapped and managed through Workflow Management Systems. The latter are generally adopted to automate all or part of business processes by flushing documents, information and tasks from one participant to another, according to a set of procedural rules and economic and organizational control principles. With specific reference to dynamic workflows, i.e. those that evolve according to the events that occur and to the ways in which tasks are handled by users, a particular problem

is represented by fairness because it is necessary to ensure that workflows have been defined adequately with regard to desired properties. To deal with this situation it is possible to use the formal verification techniques and the associated tools. The underlying idea consists in representing each business process, including interactions among human components, databases, and hardware/software elements of the information system as a finite state transition system [7], [8], [9]. In such a way, the application of formal techniques can also be relevant for strategic management decisions regarding the reliability and correctness of the design and implementation of critical business processes to achieve a competitive advantage [10] and according to the approaches of Business Process Improvement and Business Process Reengineering [11]. However, the complexity of Model Checking requires that its applicability interests those business processes where its use is justified in the light of economic considerations arising from the natural trade-off between the costs of implementation and the benefits achieved. We refer in particular to those business processes so-called business critical or commercially-critical, where the characteristics of fairness and integrity are prerequisites for the success of a market or business transaction.

In management literature there are many classifications of business processes congenial to the different purposes, fields of analysis and business sectors. In banking industry the following processes are generally identified: Management processes - that govern the operation of a system (e.g. Strategic Management, Risk Management, Internal Audit, etc.), Support processes - which support the core processes (e.g. Accounting, Human Resource, ICT, Security, Technical support, etc.), Operating processes - that constitute the core business and create the primary value stream [12].

The above-mentioned banking supply process fits typically into support processes. It is defined as the set of transactions carried out and managed by the firm to ensure the availability of resources necessary for its operation. In the banking sector, especially in Italy, the considered process suffers some problems compared to the supply processes of other sectors (industrial or service). The supply process in the bank is traditionally understood and managed as a simple business process rather than one of strategic importance i.e. banks do not seek control of the supply chain. In fact, most of the banks prepare a budget of purchases without linking it to purchase policies targeted by commodity class. In the banks there is a limited integration between the purchasing function and the logistic function also due to the confined use of typical tools of engineering procurement. Only the most advanced banking institutions have started adopting purchasing policies in collaboration with different banks to make more efficient procurement processes. Essentially, supply processes of the banks need to undertake a process of evolution towards an increasingly strategic role in analogy with what has been happening in the industry since long time [13], [14], [15].

We focus on bank supply process which fit in this context. The need to increase efficiency and reduce costs is pushing for decades many banks to review internal processes, models, organizational procedures and technological applications, in order to ensure cost-effectiveness and maintain at the same time a high level of quality of services provided to customers. We introduce the process under consideration providing a broad overview thanks to typical logic of workflow tools. The process begins whenever a bank intern subject identifies a need and formulate a request to the purchasing Manager. An evaluation of the need regarding quantity and quality follows. It is necessary to perform a search into the catalog of products in order to decide which is the right action. The purchasing Manager may also reject the request and the process terminates. Instead, if the request is approved the purchasing Manager pinpoints an adequate supplier and the purchase order is issued. Once the supplier receives the order, he prepares the goods and proceeds with the fulfillment. Afterwards, the goods reach the Head of Logistics with the invoice attachment. Here quality assurance takes place executing the related procedures. In particular, damage or non-compliance is assessed as specified in the purchase order. In case of positive evaluation, the invoice is registered with activation of the process of the accounts payable management. Finally, the goods are forwarded to whom requested them and process can complete. A detailed description of the entire workflow is shown in the flowchart of Figure 1.

B. Calculus of Communicating Systems

Historically, process algebras have developed as formal descriptions of complex computer systems, and in particular of those involving communicating, concurrently executing components. There are many examples of process algebras. Milner's Calculus of Communicating Systems (CCS) [1] is one of the most well known process algebras, and is largely used for modeling concurrent and distributed systems. Below we present only a brief overview of the main features of CCS. Readers unfamiliar with CCS are referred to [1] for further details. The syntax of *processes* is the following:

$$p ::= nil \mid x \mid \alpha.p \mid p + p \mid p|p \mid p\backslash L \mid p[f]$$

where α ranges over a finite set of actions $\mathcal{A} = \{\tau, a, \overline{a}, b, \overline{b}, ...\}$. The action $\tau \in \mathcal{A}$ is called the *internal action*. The set of *visible actions*, $\mathcal{V}$, ranged over by $l, l' \ldots$, is defined as $\mathcal{A} - \{\tau\}$. Each action $l \in \mathcal{V}$ (resp. $\bar{l} \in \mathcal{V}$) has a *complementary action* $\bar{l}$ (resp. l). The restriction set L, in the processes of the form $p\backslash L$, is a set of actions such that $L \subseteq \mathcal{V}$. The relabeling function f, in processes of the form $p[f]$, is a total function, $f : \mathcal{A} \to \mathcal{A}$, such that the constraint $f(\tau) = \tau$ is respected. The constant x ranges over a set of constant names: each constant x is defined by a constant definition $x \stackrel{\text{def}}{=} p$, where p is called the *body* of x. We denote the set of processes by $\mathcal{P}$. The standard *operational semantics* [1] is given by a relation $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$. $\longrightarrow$ is the least relation defined by the rules in Table I.

We now informally explain the semantics for CCS by induction over the structure of processes. nil represents a process that can do nothing. There is no rule for nil since it cannot evolve. The process $\alpha.p$ can perform the action α and thereby become the process p (rule **Act**). The process $p+q$ is a process that non-deterministically behaves either as p or as

TABLE I: Standard operational semantics of CCS

$$\textbf{Act}\ \frac{}{\alpha.p \xrightarrow{\alpha} p} \qquad \textbf{Sum}\ \frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad \textbf{Rel}\ \frac{p \xrightarrow{\alpha} p'}{p[f] \xrightarrow{f(\alpha)} p'[f]} \qquad \textbf{Par}\ \frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q}$$

$$\textbf{Con}\ \frac{p \xrightarrow{\alpha} p'}{x \xrightarrow{\alpha} p'}\ x \stackrel{\text{def}}{=} p \qquad \textbf{Com}\ \frac{p \xrightarrow{l} p',\ q \xrightarrow{\overline{l}} q'}{p|q \xrightarrow{\tau} p'|q'} \qquad \textbf{Res}\ \frac{p \xrightarrow{\alpha} p'}{p\backslash L \xrightarrow{\alpha} p'\backslash L}\ \alpha, \overline{\alpha} \notin L$$

q (rule **Sum** and symmetric). The operator | expresses parallel composition.

p and q may act independently: if the process p can perform α and become p', then $p|q$ can perform α and become $p'|q$, and similarly for q (rule **Par** and symmetric). Furthermore, p and q may also together engage in a communication whenever they are able to perform complementary actions. That is, if p can perform a visible action l and become p', and q can perform $\overline{l}$ and become q', then $p|q$ can perform τ and become $p'|q'$ (rule **Com**). If L is a set of visible actions, $p\backslash L$ is a process that behaves as p except that it cannot perform any of the actions (as well as the corresponding complementary actions) lying in L externally, although each pair of these complementary actions can be performed for communication internally (rule **Res**). The operator $[f]$ expresses the relabeling of actions. If p can perform α and become p', then $p[f]$ can perform $f(\alpha)$ and become $p'[f]$ (rule **Rel**). The behavior of the process x ($x \stackrel{\text{def}}{=} p$) is that of its definition p (rule **Con**).

A *(labeled) transition system* is a quadruple $T = (S, \mathcal{A}, \longrightarrow, p)$, where S is a set of states, $\mathcal{A}$ is a set of transition labels (actions), $p \in S$ is the initial state, and $\longrightarrow \subseteq S \times \mathcal{A} \times S$ is the transition relation. If $(p, \alpha, q) \in \longrightarrow$, we write $p \xrightarrow{\alpha} q$. If $\delta \in \mathcal{A}^*$ and $\delta = \alpha_1 \ldots \alpha_n, n \geq 1$, we write $p \xrightarrow{\delta} q$ to mean $p \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} q$. Moreover $p \xrightarrow{\lambda} p$, where λ is the empty sequence. Given $p \in S$, with $\mathcal{R}(p) = \{q \mid p \xrightarrow{\delta} q\}$ we denote the set of the states reachable from p by $\longrightarrow$. Given a CCS process p, the *standard transition system* for p is defined as $\mathcal{S}(p) = (\mathcal{R}(p), \mathcal{A}, \longrightarrow, p)$.

C. Language of Temporal Ordering Specification

Let us now quickly recall the main concepts of Basic LOTOS, which is similar to CCS; thus we point out only the differences with CCS. The syntax of behavior expressions is the following:

```
B ::= stop | α;B | B[]B| P | B|[S]|B |
B[f] | hide S in B | exit | B>>B | B[>B
```

where `P` ranges over a set of process names, α is an action and the internal CCS action τ is called in LOTOS `i`. As in CCS, the operational semantics of a behavior expression `B` is a labeled transition system, i.e. an automaton whose states correspond to behavior expressions (the initial state corresponds to `B`) and whose transitions (arcs) are labeled by actions in $\mathcal{A}$. The meaning of the operators composing behavior expressions is the following:

The *action prefix* `a;B` has the same meaning of the "." CCS operator, while `B1 [] B2` is similar to the CCS process $B1 + B2$ and the expression `stop` cannot perform any move, as the nil CCS operator. finally, `f` in expressions like `B[f]`, is similar to the realbelling function of CCS.

The *parallel composition* `B1|[S]|B2`, where `S` is a subset of $\mathcal{A}-\{$`i`$\}$, composes in parallel the two behaviors `B1` and `B2`. `B1` and `B2` interleave the actions not belonging to `S`, while they must synchronize at each gate in `S`. A synchronization at gate `a` is the simultaneous execution of an action `a` by both partners and produces the single event `a`. If `S`=$\emptyset$ or `S`=$\mathcal{A}$, the parallel composition means pure interleaving or complete synchronization. Cyclic behaviors are expressed by recursive process declarations. The following operators are not present in CCS:

- the *hiding* `hide S in B` renames the actions in `S`, occurring in the transition system of `B`, with the unobservable action `i`;
- the expression `exit` represents successful termination; it can be used by the enabling (`B >> B`) and disabling (`B[> B`) operators: `B >> B` represents sequentialization between `B1` and `B2` and `B[> B` models interruptions. For the sake of simplicity, we do not discuss these operators in the paper.

From now on, we write LOTOS instead of Basic LOTOS.

D. Equivalence Checking

Process algebras, like CCS and LOTOS, can be used to describe both implementations of processes and specifications of their expected behaviors. Therefore, they support the so-called single language approach to process theory, that is the approach in which a single language is used to describe both actual processes and their specifications. An important ingredient of these languages is therefore a notion of behavioral equivalence. One process description, say SYS, may describe an implementation, and another, say SPEC, may describe a specification of the expected behavior. This approach to program verification is also sometimes called implementation verification.

In the following we introduce well-known notions of behavioral equivalence which describe how processes (i.e. systems) match each other's behavior. Milner introduces strong and weak equivalences. Strong equivalence is a kind of invariant relation between process that is preserved by actions as stated by the following definition.

Definition 1 (strong equivalence): Let p and q be two processes.

- A strong bisimulation, $\mathcal{B}$, is a binary relation such that $p\ \mathcal{B}\ q$ implies: (i)

1) $p \xrightarrow{\alpha} p'$ implies $\exists q'$ such that $q \xrightarrow{\alpha} q'$ with $p' \mathcal{B} q'$; and
2) $q \xrightarrow{\alpha} q'$ implies $\exists p'$ such that $p \xrightarrow{\alpha} p'$ with $p' \mathcal{B} q'$

- p and q are strongly equivalent ($p \sim q$) iff there exists a strong bisimulation $\mathcal{B}$ containing the pair (p, q).

The idea underlying the definition of the weak equivalence is that an action of a process can now be matched by a sequence of action from the other that has the same "observational content" (i.e. ignoring internal actions) and leads to a state that is equivalent to that reached by the first process. In order to define the weak equivalence, we assume there exists a special action τ in CCS, while in LOTOS is denoted with `i`, which we interpret as a silent, internal action, and we introduce the following transition relation that ignores it. In the following we use the CCS notation to represent the silent action (τ).

Let p and q be processes. We write $p \stackrel{\epsilon}{\Longrightarrow} q$ if and only if there is a (possibly empty) sequence of τ actions that leads from p to q. (If the sequence is empty, then $p = q$.) For each action α, we write $p \stackrel{\alpha}{\Longrightarrow} q$ iff there are processes p' and q' such that:

$$p \stackrel{\epsilon}{\Longrightarrow} p' \xrightarrow{\alpha} q' \stackrel{\epsilon}{\Longrightarrow} q.$$

Thus, $p \stackrel{\alpha}{\Longrightarrow} q$ holds if p can reach q by performing an α action, possibly preceded and followed by sequences of τ actions. For each action α, we use $\widehat{\alpha}$ to stand for ϵ if $\alpha = \tau$, and for α otherwise.

Definition 2: (weak equivalence). Let p and q be two processes.

- A weak bisimulation, $\mathcal{B}$, is a binary relation such that $p \mathcal{B} q$ implies: (i)
  1) $p \xrightarrow{\alpha} p'$ implies $\exists q'$ such that $q \stackrel{\widehat{\alpha}}{\Longrightarrow} q'$ with $p' \mathcal{B} q'$; and
  2) $q \xrightarrow{\alpha} q'$ implies $\exists p'$ such that $p \stackrel{\widehat{\alpha}}{\Longrightarrow} p'$ with $p' \mathcal{B} q'$
- p and q are weakly equivalent ($p \approx q$) iff there exists a weak bisimulation $\mathcal{B}$ containing the pair (p, q).

III. Heuristic-based Methodology

Formal methods cannot be easily scaled due to the state explosion problem, which says that the state space grows exponentially in the number of concurrent processes. In fact, the parallelism among the processes of the system leads to a number of reachable states which may become very large, in some cases on the order of millions or billions of states. When the number of states is too large to fit in a computer's main memory, verification quickly breaks down. Several approaches have been developed to solve or reduce the state explosion problem. We now present an heuristic-based methodology to combat the state explosion problem for equivalence checking.

Heuristic search [3] is one of the classical techniques in Artificial Intelligence and has been applied to a wide range of problem-solving tasks including puzzles, two player games and path finding problems. A key assumption of heuristic search is that a *utility* or *cost* can be assigned to each state to guide the search by suggesting the next state to expand; in this way the most promising paths are considered first. There are several heuristic search algorithms for AND/OR graphs: a main difference among them is whether they tolerate cycles in the AND/OR graphs, or require the graphs to be acyclic. The latter class includes the AO* algorithm [16], [3], while the former includes the S2 algorithm [17]. In any case, the graph is expanded incrementally during the execution of the algorithm, starting from the initial node; an heuristic function assigns a cost to each node not yet considered.

In [4] the authors apply *heuristic search* techniques on AND/OR graphs to check equivalence of concurrent systems. Moreover, since it is sometimes better to find a solution as soon as possible, rather than to repeatedly discard solutions until the optimum is found, optimality of the solution is not really an issue from the point of view of the method efficiency. Discarding optimality, a greedy approach to the problem can be followed: at any step, the next node to expand is chosen only on the basis of the foreseen cost of its expansion. In order to apply the greedy algorithm, an heuristic function over nodes has to be defined, such function is called $\widehat{h}$ and is aimed at working during the construction of the AND/OR graph. The function $\widehat{h}$ should suggest the state containing two not bisimilar processes (w.r.t. a particular equivalence) so that this state can be included in the graph as soon as possible. The function assigns a value to each node n of the graph, called $\widehat{h}$-value of n: roughly speaking, that value is a measure of the degree of dissimilarity of the two processes in n. The heuristics function is able to manage both strong and weak equivalence and equivalently easy to be computed since they depend only on the syntax of the processes and not on their semantics. It has been defined on CCS processes. Function $\widehat{h}$ considers that a structural difference between the processes can be a good index of their non equivalence; obviously, for example, two not strongly equivalent processes, are also structurally different, and two structurally equal processes are also strongly equivalent. Nevertheless, two not structurally equal processes can be strongly equivalent; in this sense, $\widehat{h}$ is an heuristics. In this paper we use a C++ tool (Grease) implementing the approach to check both strong and weak equivalence. The tool is avaible at link: http://www2.ing.unipi.it/∼a080224/grease/. We show the effectiveness of the function $\widehat{h}$ and the heuristic-based approach using the bank supply process as case study.

A. How to Apply Equivalence Checking to the Bank Supply Process

A formal description of each component of the system must be written in order to apply equivalence checking to the Bank Supply Process. As stated above, Figure 1 depicts the interaction among the actors involved. We specified each actor as separate CCS and LOTOS processes. Precisely, we specify: Petitioner, Purchasing_Manager, Supplier, Head_of_ Logistics.

We used implementation-verification approach to perform equivalence checking. Whilst LOTOS specifications act as an input for CADP, CCS ones are used by both Grease and CWB-NC. Thus, we show either CCS or LOTOS textual descriptions in the following tables. For the sake of clarity, we limit petitioners and suppliers to one, even though we evaluated our methodology up to five and four.

TABLE II: CCS specification of the Bank Supply Process

Petitioner	$\stackrel{\text{def}}{=}$	in.('item_available.c.Petitioner1 + 'item_not_available.Petitioner1)
Petitioner1	$\stackrel{\text{def}}{=}$	'request.(proper.delivering.out.Petitioner + improper.Petitioner)
Catalog	$\stackrel{\text{def}}{=}$	item_available.'c.Catalog + item_not_available.Catalog
Purchasing_Manager	$\stackrel{\text{def}}{=}$	request.('improper.Purchasing_Manager + 'proper.'order.Purchasing_Manager)
Supplier	$\stackrel{\text{def}}{=}$	order.'invoicing.'fulfillment.Supplier
Head_of_Logistics	$\stackrel{\text{def}}{=}$	fulfillment.(compliant.'invoice_registering.'delivering.'delivering_goods.Head_of_Logistics + not_compliant.nil)
Supply_ Process	$\stackrel{\text{def}}{=}$	(Petitioner |Catalog |Purchasing_Manager |Supplier |Head_of_Logistics) \ {request,item_available,item_not_available,proper,improper,delivering,delivering_goods,order,fulfillment}

TABLE IV: LOTOS specification of the Bank Supply Process

specification P[inn,invoicing,invoice_ registering,compliant,notcompliant,outt] : noexit
beahviour hide item_ available,item_ not_ available,proper,improper,delivering,request,order,
fulfillment in ((((Petitioner[inn,item_ available,item_ not_ available,proper,improper,
delivering,request,outt]|[item_ available,item_ not_ available]|Catalog[item_ available,
item_ not_ available]) |[proper,improper,request]|Purchasing_ Manager[request,proper,improper,order])
|[order]|Supplier[order,invoice,fulfillment]) |[fulfillment,delivering]|Head_ of_ Logistics[fulfillment,
invoice_ registering,delivering,compliant,notcompliant]) **where**

process Petitioner[inn,item_ available,item_ not_ available,proper,improper,delivering,request,outt] :
noexit := inn; (item_ available; request; (proper;delivering;outt;Petitioner[inn,item_ available,
item_ not_ available,proper,improper,delivering,request,outt] [] improper;Petitioner[inn,item_ available,
item_ not_ available,proper, improper,delivering,request,outt]) [] item_ not_ available; request;
(proper;delivering;outt;Petitioner[inn, item_ available, item_ not_ available,proper,improper,
delivering,request,outt] [] improper;Petitioner[inn, item_ available,item_ not_ available,proper,
improper,delivering,request,outt]))
endproc

process Catalog[item_ available,item_ not_ available] : noexit :=
(item_ available; Catalog[item_ available,item_ not_ available] [] item_ not_ available;
Catalog[item_ available,item_ not_ available])
endproc

process Purchasing_ Manager[request,proper,improper,order] : noexit :=
request; (proper; order; Purchasing_ Manager[request,proper,improper,order] []
improper; Purchasing_ Manager[request,proper,improper,order])
endproc

process Supplier[order,invoice,fulfillment] : noexit :=
order;invoice;fulfillment;Supplier[order,invoice,fulfillment]
endproc

process Head_ of_ Logistics[fulfillment,invoice_ registering,delivering,compliant,notcompliant] : noexit :=
fulfillment;(notcompliant;stop [] compliant;invoice_ registering; delivering; Head_ of_ Logistics[fulfillment,
invoice_ registering,delivering,compliant,notcompliant])
endproc

endspec

TABLE III: CCS properties for the Bank Supply Process

P1	$\stackrel{\text{def}}{=}$	'proper.'compliant.P1 + 'improper.'not_compliant.P1
P2	$\stackrel{\text{def}}{=}$	'not_compliant.'invoice_ registering.nil
P3	$\stackrel{\text{def}}{=}$	in.'delivering_goods.PP

Note that we use an apex (i.e., 'a) in place of the bar (i.e., "$\overline{a}$") to represent output actions, as CWB-NC does. Tables II and IV represent respectively the CCS and the LOTOS specifications of the bank supply process, which are semantically equivalent. In Table IV we use the syntax of CADP to describe LOTOS processes. We chose three properties to apply implementation-verification to the considered process which we called P1 ("Proper and Compliant"), P2 ("Invoice Registering") and P3 ("Delivering Goods"). Whilst P1 means to check if a proper request always outcomes compliant goods, P2 aims to verify if it is possible to register an invoice even though the goods are not compliant. Instead, P3 check if one can obtain goods without a proper request. All the properties must be evaluated FALSE. Table III shows the definition of the properties in CCS. To perform equivalence checking, in the CCS Bank Supply Process we put as τ, every action except those syntactically occurring in the CCS properties and the ones used for communication. Afterwards weak equivalence checking is performed. Similarly, we can write the specification of the same properties in LOTOS, hiding all gates but which we are not relevant for the verification. For lack of space they are not shown. Figure 2 helps to understand interactions among the described component of the bank supply process highlighting communications.

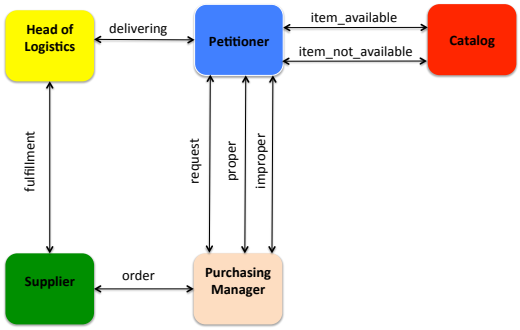

Fig. 2: Graphic representation of the interactions among all the components involved in a bank supply process.

IV. Experimental Results

In this section we show experimental results using Grease, a C++ tool to implement the methodology presented in Section III to check both strong and weak equivalence of CCS processes. Grease has been employed to verify three properties of the bank supply process: "Proper and Compliant", "Invoice Registering" and "Delivering Goods". The aim is to compare performance of our tool to CADP, a state of the art model checker. The experiments were executed on a 64bit, 2.3GHz Intel i7 quad core CPU equipped with 16 GB of RAM and running Mac OSX Mavericks 10.9.2.

Tables V, VI and VII show performace comparison regarding the aforementioned tools. Each table presents results came out checking the specified property. Column "#P#S" displays the number of "Petitioners" and "Suppliers" which we changed to obtain different implementations of the bank supply process. Columns second and third exhibit respectively Grease and CADP execution time (expressed in seconds) and memory allocation (expressed in KBytes). As can be seen looking at the tables, Grease is faster than CADP. The memory allocated by Grease grows in an exponential fashion, whilst the memory allocated by CADP grows in a linear fashion. Notwithstanding, the important speed improvement in conjunction with a modest amount of employed memory justifies the adoption of our tool when dealing with business domain, in particular bank supply processes where time is a critical matter.

V. Related Work

The most challenging task when applying automated formal verification in practice is to conquer the state explosion problem. Several approaches have been developed to solve or reduce the state explosion problem for model checking. Among them, reduction techniques based on process equivalences [18], symbolic model checking techniques [19], on-the-fly techniques [20], partial order techniques [21], compositional techniques [22], and abstraction approaches [23]. In particular, abstraction techniques have been already used to formally verify banking processes [24].

TABLE V: Grease and CADP comparison - P1

#P#S	Grease		CADP	
	Time (s)	Mem. (KB)	Time (s)	Mem.(KB)
11	0.01	1952	3.01	24112
22	0.01	2804	4.66	26176
33	0.08	12968	5.68	27394
42	0.37	14796	8.00	30100
54	23.72	233263	168.06	340164

TABLE VI: Grease and CADP comparison - P2

#P#S	Grease		CADP	
	Time (s)	Mem. (KB)	Time (s)	Mem. (KB)
11	0.01	1857	2.98	24123
22	0.01	2546	4.35	25189
33	0.08	13923	6.22	27456
42	0.41	15600	9.10	30234
54	51.58	243276	167.07	354368

TABLE VII: Grease and CADP comparison - P3

#P#S	Grease		CADP	
	Time (s)	Mem. (KB)	Time (s)	Mem. (KB)
11	0.01	1820	4.21	23600
22	0.01	2248	4.53	26084
33	0.01	3096	5.51	28632
42	0.01	3716	7.03	28892
54	0.02	7572	166.74	354300

For equivalence checking algorithms with minimal space complexity are of particular interest. Two algorithmic families can be considered to perform the equivalence checking. The first one is based on refinement principle: given an initial partition, find the coarsest partition stable with respect to the transition relation e.g. the algorithm proposed by Paige and Tarjan in [25]. The other family of algorithms is based on a Cartesian product traversal from the initial state [26], [27]. These algorithms are both applied on the whole state graph, and they require an explicit enumeration of this state space. This approach leads to the well-known state explosion problem. Classical reduction algorithms already exist [28], [29], but they can be applied only when the whole state space has been computed, which limits their interest. A possible solution is to reduce the state graph before performing the check as shown in [30], where symbolic representation of the state space is used. In [31] Lomuscio et al. presented algorithms to reduce the size of the models before the model checking step and showed preservation properties. Other algorithms are the ones by Bustan and Grumberg [32] and by Gentilini et al.[33]. For an input graph with N states, T transitions and S simulation equivalence classes, the space complexity of both algorithms is $O(S^2 + NlogS)$. The approach of Gentilini et al. represents the simulation problem as a generalised coarsest partition problem. Finally, Coons et al. [34] combined partial-order reduction with preemption-bounding to obtained a sound pruned state space.

VI. Conclusion and Future Work

The work presented in this paper shows that equivalence checking can be very useful to increase the quality in the domain of business management. In fact, using traditional equivalence checkers we find notable difficulties to prove the concreteness of real-world banking processes. This is mainly caused by the state explosion problem. We consider an efficient procedure, based on heuristic search, which uses a heuristic function that suggests to expand first the states that offer the most promising way to deduce that two systems are not equivalent. Thus, it is possible to avoid the exhaustive exploration of the global state graph of the two systems when they are not equivalent. We use Grease (GREedy Algorithm for System Equivalence), a C++ tool supporting the heuristic approach to check equivalence of CCS processes. Our investigations suggest that the business domain, in particular in the banking field, can benefit from the efficient heuristic-based methodology developed in the process algebra area to prevent significant errors. A real-world banking workflow of a supply process was chosen as a case study to demonstrate the usefulness of Grease which reduces the state explosion problem. As shown by the above experiments, our tool obtains good results when compared with a state of the art model checker CADP.

We plan to extend the number of verified properties and to automate the conversion from workflow to specification as future work. Indeed, in order to give a real benefit to the business community we need to define better tools which operate in dynamic environments [35]. Furthermore, we want to experiment our heuristic-based methodology on different processes of other domains, e.g., wikis [36], biology [37] and service composition [38].

References

[1] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989.

[2] S. Xue, B. Wu, and J. Chen, "lightepc: A formal approach for modeling personalized lightweight event-driven business process," in *Services Computing (SCC), 2013 IEEE International Conference on*, June 2013, pp. 1–8.

[3] J. Pearl, *Heuristics - intelligent search strategies for computer problem solving*, ser. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984.

[4] N. De Francesco, G. Lettieri, G. Vaglini, and A. Santone, "Heuristic search for non-equivalence checking," *ACM Trans. Softw. Eng. Methodol.*, vol. To appear.

[5] T. Bolognesi and E. Brinksma, "Introduction to the iso specification language lotos," *Computer Networks*, vol. 14, pp. 25–59, 1987.

[6] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "Cadp - a protocol validation and verification toolbox," in *CAV*, 1996, pp. 437–440.

[7] M. Kohlbacher, "The effects of process orientation on customer satisfaction, product quality and time-based performance," 29th International Conference of the Strategic Management Society, Washington DC, 2009.

[8] G. Gorry and M. S. Morton, "A framework for management information systems, vol. 13, n. 1," Sloan Management Review, 1971.

[9] R. Daft, "Organization theory and design," South Western Educ Pub., 2009.

[10] R. Kaplan and D. Norton, "The execution premium: linking strategy to operations for competitive advantage," Harward Business School Press, Boston, 2008.

[11] P. Yang, X. Xie, I. Ray, and S. Lu, "Satisfiability analysis of workflows with control-flow patterns and authorization constraints," *Services Computing, IEEE Transactions on*, vol. To Appear, 2013.

[12] I. Ognjanovic, B. Mohabbati, D. Gaevic, E. Bagheri, and M. Bokovic, "A metaheuristic approach for the configuration of business process families," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, June 2012, pp. 25–32.

[13] B. B. Flynn, B. Huo, and X. Zhao, "The impact of supply chain integration on performance: a contingency and configuration approach," *Journal of Operations Management*, vol. 28, no. 1, pp. 58–71, 2010.

[14] A. Rushton, *The handbook of logistics and distribution management*. Kogan Page Publishers, 2010.

[15] J. Cai, X. Liu, Z. Xiao, and J. Liu, "Improving supply chain performance management: A systematic approach to analyzing iterative kpi accomplishment," *Decision Support Systems*, vol. 46, no. 2, pp. 512–521, 2009.

[16] A. Mahanti and A. Bagchi, "And/or graph heuristic search methods," *J. ACM*, vol. 32, no. 1, pp. 28–51, 1985.

[17] A. Mahanti, S. Ghose, and S. K. Sadhukhan, "A framework for searching and/or graphs with cycles," *CoRR*, vol. cs.AI/0305001, 2003.

[18] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs, "Minimal model generation," in *CAV*, ser. Lecture Notes in Computer Science, E. M. Clarke and R. P. Kurshan, Eds., vol. 531. Springer, 1990, pp. 197–203.

[19] K. L. McMillan, *Symbolic model checking*. Kluwer, 1993.

[20] C. Jard and T. Jéron, "Bounded-memory algorithms for verification on-the-fly," in *CAV*, 1991, pp. 192–202.

[21] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, ser. Lecture Notes in Computer Science. Springer, 1996, vol. 1032.

[22] A. Santone, "Automatic verification of concurrent systems using a formula-based compositional approach," *Acta Inf.*, vol. 38, no. 8, pp. 531–564, 2002.

[23] D. Adalid, A. Salmerón, M. del Mar Gallardo, and P. Merino, "Using spin for automated debugging of infinite executions of java programs," *Journal of Systems and Software*, 2013.

[24] A. Santone, V. Intilangelo, and D. Raucci, "Efficient formal verification in banking processes," in *SERVICES*, 2013.

[25] R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM J. Comput.*, vol. 16, no. 6, pp. 973–989, 1987.

[26] J.-C. Fernandez and L. Mounier, ""on the fly" verification of behavioural equivalences and preorders," in *CAV*, 1991, pp. 181–191.

[27] J. Godskesen, K. Larsen, and M. Zeeberg, *TAV - tools for automatic verification: users manual*. Institute for Electronic Systems, Department of Mathematics and Computer Science, The University of Aalborg, 1989.

[28] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Sci. Comput. Program.*, vol. 13, no. 1, pp. 219–236, 1989.

[29] P. C. Kanellakis and S. A. Smolka, "Ccs expressions, finite state processes, and three problems of equivalence," *Inf. Comput.*, vol. 86, no. 1, pp. 43–68, 1990.

[30] J.-C. Fernandez, A. Kerbrat, and L. Mounier, "Symbolic equivalence checking," in *CAV*, ser. Lecture Notes in Computer Science, C. Courcoubetis, Ed., vol. 697. Springer, 1993, pp. 85–96.

[31] A. Lomuscio, W. Penczek, and H. Qu, "Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems," *Fundam. Inf.*

[32] D. Bustan and O. Grumberg, "Simulation-based minimazation," *ACM Trans. Comput. Log.*, vol. 4, no. 2, pp. 181–206, 2003.

[33] R. Gentilini, C. Piazza, and A. Policriti, "From bisimulation to simulation: Coarsest partition problems," *J. Autom. Reasoning*, vol. 31, no. 1, pp. 73–103, 2003.

[34] K. E. Coons, M. Musuvathi, and K. S. McKinley, "Bounded partial-order reduction," in *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*. ACM, 2013, pp. 833–848.

[35] G. De Ruvo and A. Santone, "An eclipse-based editor to support lotos newcomers," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2014 IEEE 23rd International Workshop on*, June 2014.

[36] G. De Ruvo and A. Santone, "A novel methodology based on formal methods for analysis and verification of wikis," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2014 IEEE 23rd International Workshop on*, June 2014.

[37] A. Santone, L. Cerulo, and M. Ceccarelli, "Infer gene regulatory networks from time series data with formal methods," *2013 IEEE International Conference on Bioinformatics and Biomedicine*, vol. 0, pp. 115–120, 2013.

[38] A. Furno and E. Zimeo, "Context-aware composition of semantic web services," *MONET*, vol. 19, no. 2, pp. 235–248, 2014.